# QuLa: Queue and Latency-aware service selection and routing in Service-Centric Networking

Piet Smet, Pieter Simoens, Bart Dhoedt

*Abstract:* **Due to an explosive growth in services running in different datacenters, there is need for service selection and routing to deliver user requests to the best service instance. In current solutions, it is generally the client that must first select a datacenter to forward the request to before an internal load-balancer of the selected datacenter can select the optimal instance. An optimal selection requires knowledge of both network and server characteristics, making clients less suitable to make this decision. Information-Centric Networking (ICN) research solved a similar selection problem for static data retrieval by integrating content delivery as a native network feature.**

**We address the selection problem for services by extending the ICN-principles for services. In this paper we present Queue and Latency (QuLa), a network-driven service selection algorithm which maps user demand to service instances, taking into account both network and server metrics. To reduce the size of service router forwarding tables, we present a statistical method to approximate an optimal load distribution with minimized router state required. Simulation results show that our statistical routing approach approximates the average system response time of source-based routing with minimized state in forwarding tables.**

*Index Terms:* **Service-Centric Networking, Information-Centric Networking, Latency-aware selection, name-based routing, QuLa**

## I. INTRODUCTION

The Internet was designed as a communications network to interconnect end-hosts and deliver data between end points in the most efficient manner. However, current Internet usage consists mostly of users retrieving the same content, which led to the development of Content Delivery Networks (CDNs). CDNs cache content in the network edge and load-balance requests over multiple replicas to reduce network latency, bandwidth and congestion. While CDNs were originally developed for static data retrieval, some CDNs (e.g. Akamai [1]) now have application delivery networks that also support data-processing applications by considering both server load and network characteristics when load-balancing. Research has shown that the Akamai CDN can significantly outperform traditional web content distribution that uses load-balancing server farms in a few datacenters [2]. Research on CDN indicates that the long-term sustainability of CDNs is jeopardized by technology heterogeneity, inefficient resource utilization, poor reactivity and coarse granularity in management operations [3].

Information-Centric Networking (ICN) [4] integrates content delivery as a native network feature and solves the selection problem by leveraging in-network caches and load-balancing. In ICN, users are able to address objects by an identifier without providing a destination locator. More specifically, users send out an anycast-like message (i.e. one identifier can address multiple replicas) to search for data, using object names instead of IP addresses to identify the desired data; it is up to the routers to forward requests to the closest data replica. This concept led to various forwarding and caching optimizations to improve content delivery [5] [6] [7].

Existing solutions to optimize content delivery, such as ICN architectures [8] [9] [10] [11] [12] and CDNs, are designed to support static data retrieval and typically do not consider complications introduced by services: caching is less evident, services are prone to dynamic service times and often require input data to consider. Currently, services often reside in datacenters or cloud sites. Cloud Computing was developed to provide easy access to computational services by facilitating resource scaling, resilience and security amongst others. At first, users communicated with only a handful of cloud sites. One of the main disadvantages of this cloud approach is the induced network latency and large bandwidth required between users and the cloud. This made a centralized cloud approach unsuitable for real-time data-processing services and motivated the development of distributed clouds located in the network edge [13] [14] [15]. Edge clouds optimize network traffic and reduce network latency by bringing services closer to the users, similar to CDNs. When several cloud sites host an instance of the same service, requests must be processed by the instance which offers the best Quality of Service (QoS) [16]. This is a more complex problem than can be solved with generalized resource assignment algorithms in individual cloud sites. Techniques to facilitate the distribution of real-time data-processing services are limited to specific cloud infrastructures; they do not focus on fine-grained selection of processing nodes in the network between the different cloud sites.

The need for an ICN-like solution to support services led to the development of Service-Centric Networking (SCN) [17]. SCN is designed to support efficient provisioning, discovery and execution of services distributed over the network. Caching in SCN is less evident because every service response is a reply to a specific request with input data. Instead, SCN architectures provide mechanisms to deploy service instances in the network and forward requests to the instance with the lowest response time. Content requests can still make use of caching for faster content delivery. Like ICN, SCN also enables location-

Piet Smet is part of the Department of Information Technology (INTEC), Ghent University - iMinds, Gaston Crommenlaan 8 bus 201, B-9050 Ghent, Belgium, email: piet.smet@intec.ugent.be.

Pieter Simoens is part of the Department of Information Technology (INTEC), Ghent University - iMinds, Gaston Crommenlaan 8 bus 201, B-9050 Ghent, Belgium, email: pieter.simoens@intec.ugent.be.

Bart Dhoedt is part of the Department of Information Technology (INTEC), Ghent University - iMinds, Gaston Crommenlaan 8 bus 201, B-9050 Ghent, Belgium, email: bart.dhoedt@intec.ugent.be.

independent access to content and services using name-based routing. SCN combines service instantiation and network routing at a fine granularity.

In this paper we present our contributions towards service selection and network routing in Service-Centric Networking. We focus on real-time data processing services which require fast system response time for an acceptable Quality of Experience (QoE). Our goal is to provide more efficient service access; users address services only by name and in-network load-balancing techniques route requests towards service instances such that the average response time as seen by the clients is optimized. We argue that network metrics such as hop-count are not sufficient for service selection and that load-balancing should be done by service routers. We aggregate user demand at network edge nodes and seek an optimal distribution of the request load across the deployed instances. Service routers forward and load-balance user demand over multiple service instances. There are two key issues to solve; (1) distributing user demand over the available service instances so that the average response time as seen by the clients is minimized and (2) configuring the forwarding tables to reflect this selection in a scalable manner.

To tackle the first issue, we present a network-driven service selection algorithm named Queue and Latency (QuLa). We implement our current approach on a centralized broker which contains knowledge of the network and server characteristics. This approach can be extended to a hierarchy of brokers for improved scalability and to avoid a single point of failure. We search for an optimal distribution of user demand over the available service instances while considering both network latency and server queuing times. The result of this step is a load distribution matrix which maps user demand to service instances.

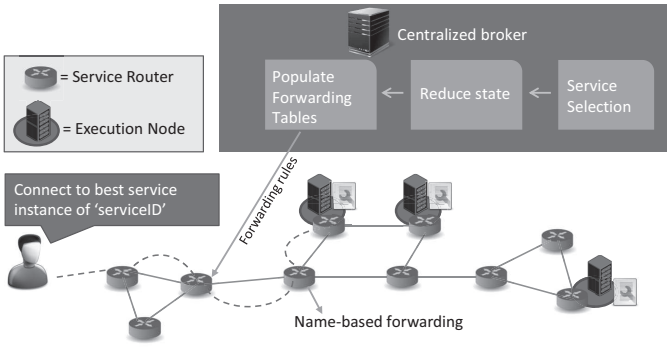



Fig. 1. A centralized broker runs the QuLa service selection algorithm, reduces the required state to map this selection to the forwarding tables of service routers and then populates these forwarding tables. Requests are forwarded to a service instance by service routers using a name-based forwarding scheme.

The second issue, populating forwarding tables, poses a scalability challenge as we envision a large amount of services and users in SCN, inducing large forwarding table sizes. Embedding the load distribution (calculated by the QuLa selection algorithm) in the forwarding tables of the service routers requires service name and client source address as input (which we refer to as source-based routing). However, source-based routing excludes the possibility of forwarding table aggregation, making this approach less practical due to poor scaling behavior. Therefore, we developed a statistical method to approximate the QuLa load distribution without considering the client source address. The result is a load distribution which tells each service router how the incoming demand should be distributed over the outgoing links, using only service name as input. However, eliminating the source address allows routing loops to occur at runtime. This is prevented by forcing requests to follow the branches of a spanning tree. We studied the impact of source address elimination on the average response time compared to source-based routing. Simulation results show that our statistical routing approach (limited by a spanning tree) with minimized router state is able to approximate the same results as source-based routing (not limited by a spanning tree). Figure 1 illustrates the different aspects of our research and an overview of the components.

The remainder of this paper is structured as follows. In section II we describe related work on ICN, SCN and name-based service selection and routing. We present our QuLa service selection algorithm which minimizes the average response time in section III. Section IV describes our proposed name-based routing approach used to load-balance requests with only service name as input. Our simulation results are presented in section V, showing that QuLa is able to approximate benchmark results with minimized router state. Finally, in section VI we present our conclusion and discuss future work.

## II. RELATED WORK

QuLa is related to several areas of research as it addresses service selection and routing in SCN. In this section we briefly discuss work relevant to QuLa for the different aspects of our design.

**Service-Centric Networking**. There are several design considerations when extending ICN to SCN, described in [17]. One of the main research challenges in SCN is setting up connections between end-hosts for service sessions without prior knowledge about the host addresses. Serval [18] proposes a Service Access Layer to translate service names to instance addresses, while simultaneously setting up the TCP connection between both end-hosts. Serval focuses on services running on mobile devices and aims to support late-binding and service migration to support seamless relocation of both users and services. SCAFFOLD [19] emphasizes on a flow-based anycast mechanism, allowing multiple instances to be addressed by the same service name. This approach relies on underlying virtualization and changes to the existing network stack.

As service selection and routing in SCAFFOLD and Serval are left open to the implementation, QuLa could be adopted to manage both tasks in these frameworks.

**Service selection**. The Zoom-in-zoom-out algorithm [20] performs server selection for a game service with certain delay constraints. The key objective is to select a minimum set of servers while still meeting the delay constraints. The algorithm starts by assigning each client to the closest server and then iteratively assigns clients to the next server on the path to the cluster center (the server with the lowest average latency to all clients). The final selection for a client is found when the latency to the next server on the path to the cluster center exceeds the latency constraints. This approach avoids overprovisioning of comput-

ing resources. This is an interesting use case for QuLa as latency constraints are the main focus for real-time services. However, this algorithm only considers network delay and does not consider the server load, making it less suitable to predict the entire response time of a service.

DONAR [21] focuses on selection for cloud services and presents a decentralized mapping scheme considering both server load and client performance. DONAR describes a problem statement which supports partial load distribution of user demand over service replicas, using a generic cost function during the selection process. QuLa adopts a similar approach and implements the cost function to predict the average response time. The service selection in QuLa will assign demand to replicas with minimum cost, reducing the average response time as seen by the clients.

In [22] service selection supports user-specific QoS in a service-oriented architecture. The selection algorithm accounts for response time, trust and monetary cost, although it does not assign jobs to resources directly. Instead, it serves as an assistant tool to recommend a number of suitable services based on the user's QoS requirements. With QuLa we avoid user interaction after the request is made by selecting the best service instance such that the average response time as seen by the clients is optimized during the forwarding process.

Selecting the best datacenter for a client is one problem, routing the response back to the user is the next challenge. In [23] both problems are addressed at the same time using a distributed algorithm based on alternating direction method of multipliers (ADMM), minimizing cost (bandwidth, electricity) and maximizing performance (latency). However, this algorithm uses end-to-end propagation delay as performance measurement and assumes that link capacity is more restricting than server processing capacity. In SCN, servers are placed at the network edge on execution points which are likely to be smaller and less powerful than large datacenters; this requires less bandwidth and increases the importance of server processing capacity. In QuLa we solve the selection and routing problem separately while considering both network and server characteristics.

**Name-based Routing**. In IP anycast, one IP address can correspond to several service replicas. However, native IP anycast redirects traffic to destinations based on the shortest path and does not consider server metrics. A proposal was made to extend IP anycast to a load-aware anycast CDN [24], where a centralized controller considers both network and server load to drive the CDN redirection mechanism. An interesting aspect of this research is the focus on minimizing the traffic disruption when ongoing sessions are being re-mapped to alternative CDN servers. Unfortunately, this approach is only feasible if the Autonomous Systems (ASes) it targets have a large footprint in the country where they provide CDN services. Also, this approach is focused on anycast routing in CDNs, which does not overcome the limitations of CDN itself.

Akamai, one of the larger players on the CDN market, avoids network hotspots by using its extensive network and server monitoring to redirect clients to frequently changing Akamai edge servers, lowering the client-perceived latency. It is shown that using an Akamai-server as a one-hop detour (client to Akamai edge server to destination) is more beneficial than using the direct client-destination path in more than 50% of the scenarios [2]. This shows how traditional IP routing is not always optimal for traffic with low latency requirements. Therefore, we adopt the forwarding scheme of the ICN framework Content Centric Network / Named Data Networking (CCN/NDN) [8]; each router forwards requests to the next hop on path to the desired data. This allows separate paths to be set up for different services, even if multiple services run on the same destination. When hotspots are detected, the traffic distribution over the outgoing links of service routers can be adjusted to steer traffic away from the hotspots and improve overall performance.

In ICN, routers typically forward requests based on object IDs rather than destination IP address. *One approach* is to extend existing routing protocols for named-based requests, such as OSPF-N [25]. This approach extends the OSPF link-state routing protocol for IP networks to support name-based requests. *A second approach* is to develop new forwarding schemes to learn which interfaces requests should be forwarded on, such as Greedy Ant Colony Forwarding (GACF) [26]. The GACF forwarding algorithm uses Ant Colony Optimization, a probabilistic optimization heuristic, to find the best paths to forward requests on. However, this approach does not consider server characteristics (e.g. load), rendering it less suited for SCN. *A third approach* is to extend existing ICN platforms with routing protocols aimed to provide service access. One of these approaches is SoCCeR [27], a decentralized routing protocol for services built on top of CCNx (an implementation of CCN). Routing in SoCCeR uses Ant Colony Optimization to gather latency and service load information which is used to configure the Forwarding Information Base of CCNx nodes. SoCCeR provides a plausible way to learn which instances provide the fastest response time through continuous learning at runtime. Rather than adjusting load-balancing probabilities at runtime, QuLa finds an optimal distribution given a certain demand and service placement, minimizing the activity of the routing plane.

In CCN/NDN, requests can be forwarded to several destinations while only the first answer is accepted. As services generally consume more computational resources than data retrieval, service selection is desired to provide one-to-one mapping between service requests and service instances. Therefore, the routing process presented in this paper is a combined effort of the QuLa selection algorithm, which selects several acceptable destinations for service requests of each user, and the service routers which perform statistical load-balancing (based on service name) to route each request to one of these destinations in a hop-based forwarding manner.

## III. QULA: NETWORK-DRIVEN SERVICE SELECTION

One of the key issues presented in this paper is to map user demand to available service instances such that the average response time is minimized. In ICN, routers must find a path to a location-independent name. In most standardized routing protocols, path selection is often based on network metrics (hop-count, bandwidth, network latency ...) used in shortest path algorithms such as Dijkstra's algorithm [28]. We argue that service selection in a service-centric network (1) should consider

both network and server characteristics and (2) benefits from monitoring traffic patterns in real-time.

To address the first concern, we describe a problem statement and present an objective function which considers both server and network metrics to minimize the average response time as perceived by the clients. In our model, each server has a queue for incoming requests when the server is busy. We use the queue size to represent the server load and consider latency as network metric, which is why our objective function is called Queue and Latency (QuLa). We implemented Simulated Annealing [29] to perform service selection using the QuLa objective function and compare this approach with (1) a greedy shortest path approach, (2) assigning equal load to each server and (3) a dynamic assignment of each request to the shortest queue upon request arrival.

The second concern is studied in section V-F where we show the impact of performing service selection with less frequent monitoring.

### A. ASSUMPTIONS

The service selection algorithm maps user demand to available service instances. However, considering every user individually is not feasible when load-balancing must be done in real-time. Therefore, we aggregate the load generated by a group of users, located in a nearby geographical area, into an aggregated load from client node *i*. In the remainder of this paper we use the term 'client node' to refer to a node from which demand is generated that reflects the aggregated demand of a group of nearby users. This allows us to reduce the size of the forwarding tables in the service routers. The requests from all users in the geographical area represented by client node *i* are forwarded by the service routers according to the forwarding rules set for client node *i*. A server node *j* represents a collection of computing resources located in a nearby geographical area (e.g. datacenter or cloud site), with a queue for incoming requests.

Next, we assume a fixed service placement and fixed user demand in time; service instances are not migrated, added or removed during experiments, and users may have different request rates but these do not change over time. We made these assumptions as we wish to evaluate the performance of the initial solution. Changing service placement or user demand requires redistribution of load, which can be solved by simply performing another run of the selection algorithm based on the new conditions. The importance of varying request rate is investigated in section V-F.

Last, we assume that service processing times are reproducible and stable, and that client nodes are implemented such that requests arrive with an average rate according to a Poisson process. Therefore, we use M/G/1 queue to model the server queuing time. If these conditions are not fulfilled, the same approach as described in this paper can be adopted for other queuing systems. Using an M/G/1 queue, datacenters running several service instances can be modeled in QuLa by separate servers with zero link delay to a common node, each running one service instance.

### B. PROBLEM STATEMENT

Consider a network graph containing edges *E*, nodes *N* and services S. The lambda values $\lambda(i,s)$ represent the request rate from node *i* for service *s*. Client nodes are denoted by $N_c \subset N$. Nodes hosting at least one instance of a service $s \in S$ belong to the server nodes $N_S \subset N$. The load distribution matrix ***R(i,j,s)*** $\in [0,1]$ denotes the fraction of load from client node *i* for service *s*, to be processed on server node *j*. $\overline{T_{j,s}}$ is the average service time to process a request for service *s* on server node *j*, not considering queue delay.

A generic objective to map user demand to service instances, using demand fractions, is the following:

$$\min \sum_{i \in N_c} \sum_{j \in N_S} \sum_{s \in S} cost(i,j,s) * R(i,j,s) * \lambda(i,s) \quad (1)$$

The objective in the current implementation is to optimize the average response time given a fixed service placement and fixed user demand. There are two major factors that affect the response time of a service; the time spent in the network and the time spent on the server. Servers processing a larger demand have more impact on the average response time. Taking into account the above, we find the following objective:

$$\min \frac{\sum_{i \in N_c} \sum_{j \in N_S} \sum_{s \in S} (T_{Lat.} + T_{proc.}) * R(i,j,s) * \lambda(i,s)}{\sum_{i \in N_c} \sum_{s \in S} \lambda(i,s)} \quad (2)$$

The sum of $T_{Lat.}$ and $T_{proc.}$ represents the response time of a single request, considering the network latency and the estimated time spent on the server. The product of $R(i,j,s)$ and $\lambda(i,s)$ denotes the contribution to the average response time when sending the fraction $R(i,j,s)$ of demand $\lambda(i,s)$ to the service *s*. Finally, we normalize the numerator by dividing by the total user demand to get the response time, used as quality representation for service selection. $T_{Lat.}$ is the Round Trip Time (RTT) between the client node *i* and server node *j*. $T_{proc.} = f(R(i,j,s))$ denotes the time spent on the server, including queue delays and service time.

We illustrate this with an example assuming that our system is an M/G/1 queuing system; i.e. $T_{proc.} = \frac{\left(\lambda * \overline{T_{j,s}^2}\right)}{2*(1-\rho)} + \overline{T_{j,s}}$ (Pollaczek-Khinchin mean value formula) which is the sum of the average queue delay and the average service time. $\rho$ denotes the total incoming request rate on node *j* divided by the service rate. Considering the influence of $R(i,j,s)$ we find $\rho = \overline{T_{j,s}} * \sum_{i \in N_c} \lambda(i,s) * R(i,j,s)$ with $1/\overline{T_{j,s}}$ being the service rate.

To guarantee that the demand of each client node is completely satisfied, the objective is limited by the following constraint:

$$\forall\, i \in N_c,\ \forall\, s \in S : \sum_{j \in N_S} R(i,j,s) = 1 \quad (3)$$

Using Equation 2 as objective function, we now search for an optimal load distribution to optimize the average response time. However, the distribution matrix R in the objective function takes floating point numbers as element values, creating an

| Parameter | Value | Description |
|---|---|---|
| T | 10 000 | The temperature decides the likelihood of accepting a solution worse than the current best one. At higher temperatures Simulated Annealing is more likely to accept a worse solution to continue exploring the search space. |
| $T_{stop}$ | 1 | The temperature at which Simulated Annealing stops exploring the search space and returns the best found solution. |
| repetitionCount | 2 | This variable determines how many solutions are explored at one temperature value. |
| coolingRate | 0.01 | The speed at which the temperature decreases. |
| Delta ($\Delta$) | 0.1 | Indicates the amount of change made to a solution when generating neighboring solutions. |

Table 1. parameters used for *SA*

infinitely large solution space. In the following sections we describe several alternative algorithms to find a load distribution matrix R.

**Algorithm *SA*** Simulated Annealing

**Input:** Temperature T, repetitionCount, coolingRate
**Output:** Solution $S_{best}$ with highest energy $E_{best}$

$S_{current} \leftarrow generateSolution()$
$S_{best} \leftarrow S_{current}$
$E_{best} \leftarrow E_{current} \leftarrow Objective(S_{current})$
**while** $T > T_{stop}$ **do**
  **for** $1 \rightarrow repetitionCount$ **do**
    $S_{new} \leftarrow createNeighborSolution(S_{current})$
    $E_{new} \leftarrow Objective(S_{new})$
    /* Generate random $\in [0, 1[$ and calculate probability to accept worse solution */
    **if** $acceptanceProbability(E_{current}, E_{new}, T) > Random()$ **then**
      $S_{current} \leftarrow S_{new}$
      $E_{current} \leftarrow E_{new}$
    **end if**
    **if** $E_{current} > E_{best}$ **then**
      $S_{best} \leftarrow S_{current}$
      $E_{best} \leftarrow E_{current}$
    **end if**
  **end for**
  $T \leftarrow T * (1 - coolingRate)$
**end while**

### C. SIMULATED ANNEALING

We implemented Simulated Annealing (SA) to take into account both server load and network latency during service selection. SA is a search heuristic to explore a large solution space in a short timeframe, inspecting multiple local minima in the solution space, but does not guarantee to find an optimal solution in a finite amount of time. At high temperature values SA is likely to accept a solution worse than the current best and covers a large search space by escaping from local minima. When the temperature lowers, the probability of accepting worse solutions also decreases and SA starts focusing on a smaller search space around the current best solution. Initial temperature values are usually set high to allow the algorithm to explore any solution before it starts inspecting local minima.

We performed a parameter sweep to evaluate starting temperatures between T=[100,20000] with a step of 200. We concluded that the starting temperature T has very little influence on the quality of the final solution for our problem, but the calculation time is reduced by approximately 30% when the starting temperature is reduced by a factor 10. For the topologies used in our simulations we found a good solution with a starting temperature of T = 10000. Since our static selection algorithm does not need to run frequently, it is more important to find a good and stable solution rather than reducing the execution time. In our simulations the execution time of *SA* was between 1 second for our smallest topology and 60 seconds for our largest topology. We discuss our simulation setup and topologies in section V-A.

The repetition count allows the annealing process to evaluate several solutions at the same temperature, which covers a larger search space but also increases execution time. For a more detailed explanation of Simulated Annealing we refer to [29]. The pseudo code of SA is shown in Algorithm *SA* and the parameters used are described in Table 1.

In the following paragraph we present our implementation of the significant steps in the Simulated Annealing process.

*GenerateSolution* is used to generate an initial solution to start exploring the search space. In section III-D we describe two alternative algorithms which can be implemented in *generateSolution* to construct a solution although many approaches are supported. All generated solutions must adhere to Equation 3.

**Algorithm** createNeighborSolution()

**Input:** $\Delta$, currentSolution
**Output:** Slightly altered load distribution matrix $R$

$R \leftarrow currentSolution$
$i \leftarrow$ pick random client $\in N_c$
**for all** $s \in S$ **do**
  $x \leftarrow$ pick random server $\in N_S$
  $R(i, x, s) \leftarrow R(i, x, s) + \Delta$
  **for all** $j \in N_S$ **do**
    $R(i, j, s) \leftarrow R(i, j, s)/(1 + \Delta)$
  **end for**
**end for**

For each iteration of *SA* we generate a new solution to further

explore the search space. These new solutions are generated by making changes to the current solution; we refer to them as *neighboring solutions*. In our implementation neighboring solutions are generated by assigning more demand to one server and removing the same amount from the remaining servers (cfr. Equation 3).

We opted for $\Delta = 0.1$ to explore a large search space in a short timeframe, with relatively small deviation between neighboring solutions.

After creating a neighboring solution, *SA* must accept or reject the new candidate solution. A candidate solution is always accepted if it is better than the current solution. However, to escape local optima, a candidate solution worse than the current solution can be accepted using the acceptance probability:

$$e^{\frac{(CurrentEnergy\ -\ newEnergy)}{T}} \tag{4}$$

where the energy represents the response time calculated with Equation 2. Once either of the stop conditions is met, we return the best solution encountered during the annealing process.

### D. BENCHMARK ALGORITHMS

In this section we describe two alternative approaches to find a load distribution, which are then used as benchmarks to evaluate *SA*.

**1. Greedy Algorithm**. We implement a greedy load distribution algorithm which prioritizes client-server pairs with the smallest latency. Starting with the client-server pair inducing the smallest network latency, user demand is assigned to that server until either the client node's demand is completely satisfied, or the server capacity (maximum amount of requests processed per time unit) is met. We iterate through client-server pairs until all client node demand is satisfied.

**Algorithm *Greedy*** assign demand by prioritizing client-server pairs with the lowest latency

**Input:** $\lambda(i,s)$, $\overline{T_{j,s}}$ $\forall i,j,s$
**Output:** Load distribution matrix $R$

$A(i,s) \leftarrow \lambda(i,s),\ \forall i,s$
$C(j,s) \leftarrow 1/\overline{T_{j,s}},\ \forall j,s$
*/ sort all (i,j,s)-triplets by increasing network latency between i,j */
$P \leftarrow sort\,(i,j,s)$
**for all** $i,j,s\ in\ P$ **do**
  **if** $A(i,s) > 0$ **and** $C(j,s) > 0$ **then**
    **if** $A(i,s)/C(j,s) > 1$ **then**
      $A(i,s) \leftarrow A(i,s) - C(j,s)$
      $R(i,j,s) \leftarrow C(j,s)/\lambda(i,s)$
      $C(j,s) \leftarrow 0$
    **else**
      $C(j,s) \leftarrow C(j,s) - A(i,s)$
      $R(i,j,s) \leftarrow A(i,s)/\lambda(i,s)$
      $A(i,s) \leftarrow 0$
    **end if**
  **end if**
**end for**

This approach is very intuitive, assigning most demand to the client-server pairs which induce the lowest response time. We used this greedy distribution as starting solution for *SA* in our simulations. However, by prioritizing a certain client-server pair, we indirectly penalize other clients which do not get to use this server's full capacity anymore. This implies that a non-greedy decision for client-server pairs with low latency can have positive effects on the response time of several nearby clients. In section V we present our results on this hypothesis.

**2. Equal Share.** *Greedy* always induces high workload on few servers while more distant servers remain idle most of the time. An alternative is to give each server an equal share of user demand (*Equal*), trading reduced workload for increased network latency.

$$R(i,j,s) = \frac{1}{size\,(N_S)}\ \forall\, i \in N_c,\ j \in N_S,\ s \in S \tag{5}$$

In section V we show that this approach is less sensitive to increasing demand although it does induce more network latency.

**3. Joint Shortest Queue (JSQ).** The disadvantage of static selection algorithms is the poor resilience to unexpected load conditions. Small peaks could temporarily render a server less suitable for request processing. When the service routers only follow the static forwarding table configuration without considering actual server load upon request arrival, the performance of that server could further decrease and impact the overall response time.

Dynamic selection algorithms can mitigate this problem by assigning requests to servers upon request arrival, based on measured metrics. However, this requires monitoring information to be available when assigning a request to a server (in our case, on the service routers). Gathering both network and server load information on each service router is difficult to scale which is why most dynamic selection algorithms only consider server metrics or network metrics.

To evaluate the performance difference between a static configuration and a dynamic selection algorithm, we implement Joint Shortest Queue (*JSQ*), an often used selection algorithm for server farms [30]. Upon request arrival, JSQ assigns each request to the server with the least number of unfinished requests to minimize queue delays. This approach enables the system to minimize the queue length and the mean response time on every server [31].

However, unlike Equation 2, JSQ does not consider the network latencies between client and server. In section V-D we study the performance difference between a onetime static configuration by solving Equation 2 with *SA*, and a dynamic selection with *JSQ*.

## IV. CONFIGURING THE NAME-BASED ROUTING PLANE

When demand is high, user requests must be load-balanced over multiple service instances. This is reflected in the selection algorithm by using a load distribution matrix *R*, assigning partial demand to available instances. At runtime, service routers load-balance user demand utilizing the fractions assigned in the

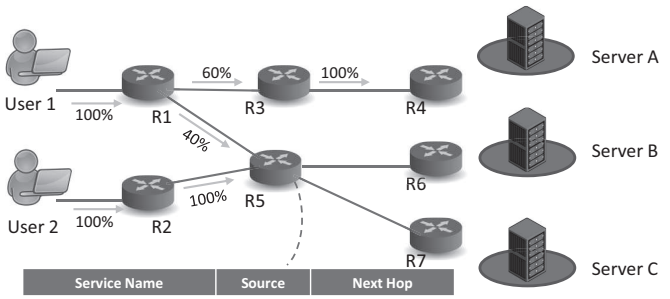
load distribution matrix. In this section we describe the configuration process of statistical load-balancing in the service routers to approximate the service selection.

We build on top of the hop-based forwarding scheme from CCNx where each router knows only the address of the next hop towards the service instance. Unlike the CCNx approach, service selection prevents requests from being processed by multiple service instances as only one answer is accepted by the client. For a perfect mapping of the service selection to the forwarding tables, each service router should also consider the source of the request (*source-based routing*). This approach is illustrated in Figure 2. As this inflates the forwarding tables, we seek to approximate the same results without considering the source of a request. To tackle this problem, we propose a statistical load-balancing method which is performed by each service router on the path to a service instance. An important research question is how much this approximation degrades the overall average response time. In the next section we describe both variants and we present our results in section V-C.

### A. VARIANT 1: SOURCE-BASED ROUTING

We configure service routers with separate forwarding entries for each source address to reflect the service selection. Consider $\lambda(i,s)$ the total demand from client *i* for service *s*, $R(i,j,s)$ the percentage of that demand to be processed on server *j*, $P_k^{in}(i,s)$ the incoming percentage of $\lambda(i,s)$ on service router *k*, and $P_{kl}^{out}(i,s)$ the percentage of $P_k^{in}(i,s)$ forwarded to router *l* on router *k*. Initially, all $P^{in}$ and $P^{out}$ are set to zero. The forwarding tables are configured as follows: (1) we stipulate a path for each client-server pair (*i,j*) and a given service *s*. (2) For each router *k* on that path, $R(i,j,s)$ is added to both $P_k^{in}(i,s)$ and $P_{kl}^{out}(i,s)$, where *l* is the next service router on path. (3) After all pairs *(i,j)* are traversed for service *s*, we express $P_{kl}^{out}(i,s)$ as fraction of $P_k^{in}(i,s)$, to normalize all values in range [0,1].

Figure 2 shows a sample configuration: 40% of user 1's demand is sent to R5, which forwards 50% to R6 and 50% to R7. Thus, 20% of user 1's demand reaches zone B and 20% reaches zone C, as per selection.




| Service Name | Source | Next Hop |
|---|---|---|
| Serv-facedetection | U1 | R6 (50%), R7 (50%) |
| Serv-facedetection | U2 | R6 (20%), R7 (80%) |

Fig. 2. source-based routing enables accurate load-balancing but is prone to large forwarding tables

To populate the forwarding table of service router *k* we distribute $P_{kl}^{out}(i,s)$ for each pair (i,s) and each neighbor router *l*.

### B. VARIANT 2: WEIGHTED AVERAGE

An ideal mapping of the load distribution matrix $R(i,j,s)$ to the forwarding tables of service routers requires a source address. To reduce the forwarding table sizes, we approximate the average response time of source-based routing with statistical load-balancing on service routers, without considering the source of the request. The following paragraph describes our statistical method to approximate the load distribution from the selection result without considering source addresses, executed on a centralized broker which contains the entire distribution matrix. The outcome of this method, a load distribution which does not contain source addresses, is then distributed to the service routers.

Averaging the outgoing percentages $\sum_{i \in N_c} P_{kl}^{out}(i,s)$ does not suffice to reflect the service selection. We observe that outgoing demand on a service router *k* is influenced by both larger $\lambda(i,s)$ and $P_k^{in}(i,s)$. Using the same configuration steps as the source-based routing variant, we find the total incoming demand for service *s* on router *k*, $D_k^{in}(i,s) = \sum_{i \in N_c} \lambda(i,s) * P_k^{in}(i,s)$ , and the outgoing demand on the link to router *l*, $D_{kl}^{out}(i,s) = \sum_{i \in N_c} \lambda(i,s) * P_k^{in}(i,s) * P_{kl}^{out}(i,s)$. We approximate

$$P_{kl}^{out}(s) = \frac{D_{kl}^{out}(s)}{D_k^{in}(s)} = \frac{\sum_{i \in N_c} \lambda(i,s) * P_k^{in}(i,s) * P_{kl}^{out}(i,s)}{\sum_{i \in N_c} \lambda(i,s) * P_k^{in}(i,s)} \quad (6)$$

where $P_{kl}^{out}(s)$ denotes the new outgoing percentages using only service name as input and not taking into account the source of the request. Only $P_{kl}^{out}(s)$ is configured in the forwarding tables, enabling service routers to load-balance requests as stated per service selection without considering source address. $P_{kl}^{out}(s)$ approximates the same amount of traffic forwarded on each edge as the source-based routing variant, inducing approximately the same system response time.

The logic behind this method goes as follows: assuming source-based routing, we calculate the incoming and outgoing traffic load on every service router for all clients. Equation 6 is used to find the outgoing traffic distribution for every edge on a service router to approximate the same traffic load without considering source addresses. When the amount of traffic on each edge approximates the load induced with source-based routing, each server also receives approximately the same load as stated by source-based routing. Due to service routers not considering source addresses, it is possible for the actual load distribution to deviate from the client-server mapping found by the algorithms described in section III. Now consider a client *i* inducing more load on server *j* than stated by the load distribution $R(i,j,s)$ because of our weighted average load-balancing. If the server load remains unchanged, there must be another client *k* inducing less load on server *j* and more load on another server *m* than stated by $R(i,j,s)$. Due to the load-balancing performed on each service router, requests are less likely to reach more distant servers. Thus, an increased load on server *j* indicates that server *j* is likely to be located nearby client *i*, improving the average response time for client *i*. Client *k* may now see a decrease or increase in average response time depending on the location of server *m*. However, not all clients can benefit from this distribu-

tion shift or else $R(i,j,s)$ would not be an optimal load distribution. These clients experience an increased response time and negate the improved response time of other clients (e.g. client *i*), approximating the average system response time as predicted with Equation 2. This effect is studied in section V-E.

Due to service routers load-balancing requests without considering source addresses in QuLa weighted average routing, routing loops may occur if no further action is taken. Therefore, allowed routing paths should be restricted to edges belonging to a minimum spanning tree of the network graph. A minimum spanning tree of a network graph is a tree that contains every vertex of the graph, where the total weight of all the edges is minimized. To construct a spanning tree we use Kruskal's algorithm [32]. By using a suitable metric as edge weight in Kruskal's algorithm to construct a spanning tree, we can prioritize edges which contribute most to the response time, reducing the performance loss compared to using the full network graph. In section V-E we study the effect of different edge metrics in Kruskal's algorithm on the average response time, propose a metric derived from the load distribution and present our simulation results. All results for weighted average routing in section V-C are obtained by forcing requests to follow the branches of a minimum spanning tree.

## V. SIMULATION RESULTS

We evaluate the service selection algorithms from section III in a simulator using several sample network topologies to determine the most efficient selection approach. In section V-A we describe our simulation setup and network characteristics, followed by a discussion of the service selection performance in section V-B. Next, we evaluate the two routing variants and the degradation in average response time induced by QuLa weighted average routing in section V-C. In order to determine the performance difference of a onetime static selection approach and a dynamic selection algorithm, we evaluate the impact of performing selection upon request arrival but with less information than the static approach in section V-D. Not considering source-addresses allows routing loops to occur, which is solved by using a spanning tree to determine allowed routes. In section V-E we study the importance of the metric used to prioritize edges in Kruskal's algorithm and its impact on the performance of QuLa's weighted average routing. We conclude our evaluation by studying the necessity of demand monitoring and dynamic adaptation in section V-F.

### A. SIMULATION SETUP

In order to evaluate large network topologies we created a simulation environment using CloudSim [33], a framework for modeling and simulating cloud computing infrastructures and services. We extended CloudSim's datacenter objects to add an internal service router and use each datacenter to host one service instance per virtual machine. This approach avoids several service instances affecting the service time on one virtual machine. In our setup all servers are modelled by an M/D/1 queue, a special case of M/G/1 with deterministic service time ($\overline{T_{j,s}^2} = \overline{T_{j,s}}^2$). In this case the server processing time described in section III-B can be written as $T_{proc.} = \frac{(2-\rho)}{2*(1-\rho)} * \overline{T_{j,s}}$. In CloudSim an M/D/1 queue system is implemented by (1) scheduling requests using CloudSim's Space Shared Scheduler, (2) configuring components so that each request has a fixed service time, and (3) implementing a Poisson process on client applications.

| | SS50 | SD50 | LS200 | LD200 |
|---|---|---|---|---|
| #Service routers | 50 | 50 | 200 | 200 |
| #Clients | 5 | 5 | 20 | 20 |
| #Servers | 3 | 3 | 3 | 3 |
| Outgoing edge degree per router | 1 | 5 | 1 | 5 |
| Avg diameter | 12 | 6 | 19 | 8 |
| Service time | 25ms | | | |
| Link latency distribution | uniform [10,100] | | | |
| Brite Model | Waxman AS | | | |

Table 2. network configuration of each topology type used in simulations.

In order to evaluate the service selection and routing behavior in different network environments, we generate four types of network topologies using Brite [34] (Table 2): (1) small topologies with sparse connectivity (SS50), (2) small topologies with dense connectivity (SD50), (3) large topologies with sparse connectivity (LS200) and (4) large topologies with dense connectivity (LD200). For each topology type we generate 50 sample networks, assign client-server roles, place service replicas, and generate user demand patterns. Using the generated topologies and fixed service placement, service selection is performed by the algorithms described in section III which return the load distribution matrix. We implemented a centralized component, the broker, to configure service routers residing in each datacenter based on this load distribution matrix. In the final configuration step, the broker deploys the service process on each server node, ready to process requests. Using the average response time of these simulations we obtain reliable data required to make a confident assessment of the performance of each algorithm. All simulations are performed on the iLab.t Virtual Wall [35] using a server with a Hexacore Intel E5645 (2.4GHz) CPU, 24GB RAM, 1x 250GB hard disk and 1-5 gigabit network interface cards.

### B. SERVICE SELECTION PERFORMANCE

To compare *SA*, *Greedy* and *Equal*, Equation 2 is used to calculate the expected response time for each solution. Figure 3 illustrates the performance of each algorithm, represented by the (expected) average response time in milliseconds (Y-axis) for a set of fixed load values (X-axis). Response times on the graphs represent the average response time for each topology type, obtained by sampling 50 topologies for fixed load values.

Consider Figure 3, for low system load the response time is dominated by network latency, making *Greedy* an optimal solution as it prioritizes servers closest to the users. *Equal* does not consider server location and network latency which renders this solution less suitable for varying network latencies between

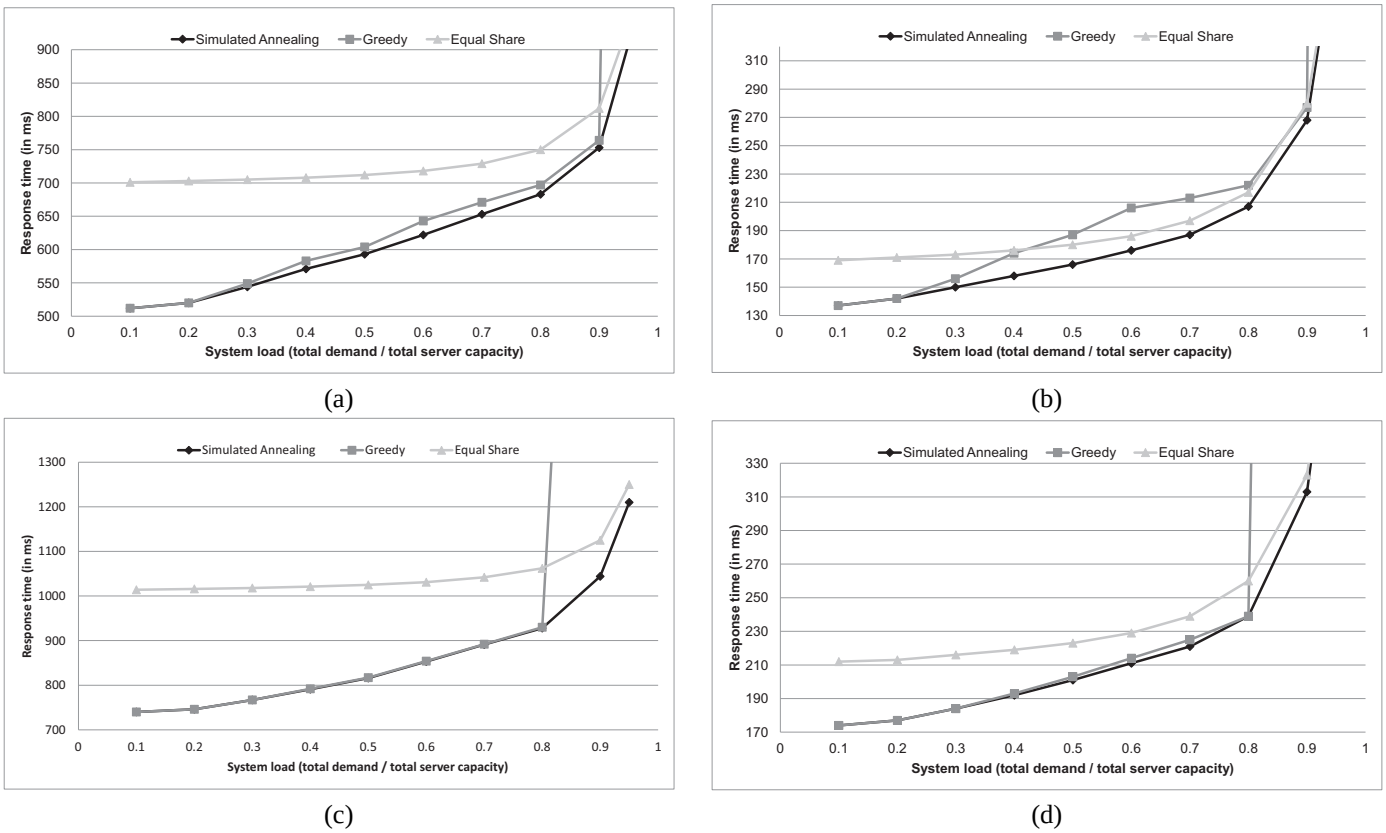



Fig. 3. comparison of the service selection solution quality for (a) a small topology with sparse connectivity, (b) a small topology with dense connectivity, (c) a large topology with sparse connectivity and (d) a large topology with dense connectivity.

client-server couples. The service selection in *Greedy* will prioritize the closest server until its maximum capacity is reached before assigning requests to other servers. These servers operating near maximum capacity are the reason for the poor performance of *Greedy* for high demand (at 90% in figure 3 a and b, at 80% for c and d). *SA* is able to adapt to increasing load by shifting away from a greedy approach and distribute load over more server replicas, similar to *Equal*. When using these approaches instead of *Greedy*, servers only operate near maximum load when the total system load approaches 95%. This increases system stability and allows for higher load values to be processed on the network.

For average system load we observe that both *SA* and *Equal* perform better in dense graphs (Figure 3 b and d) than sparse graphs (Figure 3 a and c). This is due to dense graphs containing more paths between the different client-server couples, lowering the average hop count and reducing the penalty of not considering network latency in *Equal*. *SA* performs better in dense graphs due to the network containing more available paths and thus more load distribution solutions possible. A sparse graph contains fewer and longer paths between client-server pairs, increasing the latency penalty when forwarding requests to more distant servers. This makes *Greedy* an efficient and near-optimal approach for networks with sparse connectivity as it assigns as much demand as possible to the nearest servers. In a sparse network we observe a slightly better performance with *SA*, although one could argue that the performance gained is not worth the additional computing time and resources. Despite the increased execution time, this approach still has the benefit of supporting more system load than *Greedy* in both dense and sparse networks.

### C. QULA WEIGHTED AVERAGE VS. SOURCE-BASED ROUTING

We investigate to what extent the QuLa weighted average approach (section IV-B) is able to approximate the response time achieved through source-based routing (section IV-A), without maintaining large source state in forwarding tables. To avoid routing loops, weighted average routing was simulated on a minimized spanning tree as described in section IV-B.

Figure 4 shows the measured response times using source-based routing (dashed) and the QuLa weighted average approach (solid) to configure the forwarding tables based on the service selection obtained by *SA*, *Greedy* and *Equal*. Source-based routing guarantees an exact mapping of the load distribution into the routing configuration. Therefore, the response times obtained through source-based routing simulations correspond to the expected (theoretical) values illustrated in Figure 3 and are used as a benchmark for our weighted average approach.

For low load values *Equal* induces a larger response time than *SA* and *Greedy* as explained in section V-B. However, we observe that the performance of both source-based routing and QuLa weighted average routing depends on the network connectivity (sparse or dense). There are two key factors that contribute to this phenomenon. First, in *Equal* users send equal

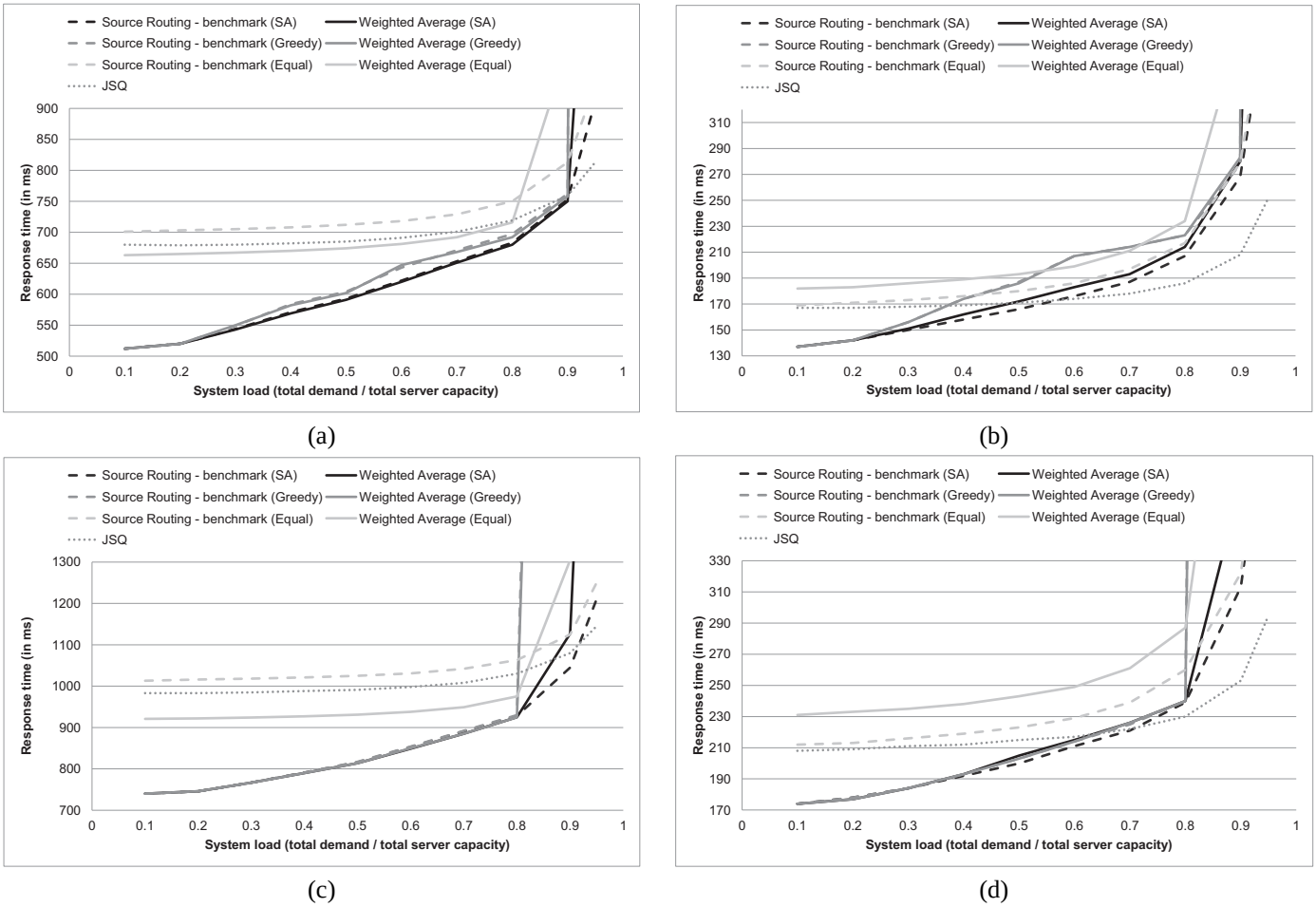


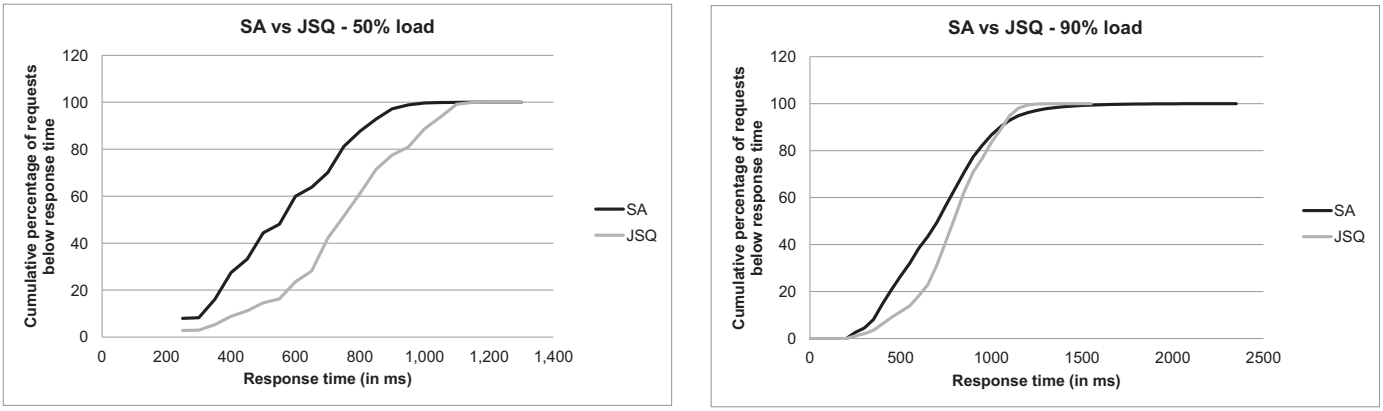

Fig. 4. response times using source-based routing and weighted average for (a) a small topology with sparse connectivity, (b) a small topology with dense connectivity, (c) a large topology with sparse connectivity and (d) a large topology with dense connectivity.

load to each server using many edges in the network, while the spanning tree used in QuLa weighted average routing only contains a subset of those edges. For every edge no longer available in the spanning tree, requests are forced to take a longer route than expected and the response time increases. Second, service routers in QuLa's weighted average routing perform statistical load-balancing; service routers can forward requests to a local server and the probability of requests reaching more distant servers decreases as more service routers are passed. The spanning tree in QuLa weighted average routing contains longer and fewer paths so that requests are less likely to reach distant servers, similar to a greedy service selection. This reduces the penalty of using a spanning tree as the average network latency decreases when using QuLa weighted average. We describe this effect in detail in section V-E.

Depending on the network characteristics, one of these two effects will outweigh the other. *Dense networks* contain many different paths between two nodes and traffic is spread out over several paths, inducing low traffic load on individual edges. As the spanning tree only contains a small set of the most used edges, the traffic must now follow longer paths than before and thus the average network latency increases. *In sparse networks* the penalty of losing edges by using a spanning tree is minimized as the traffic is already concentrated on the few available paths. This allows construction of a spanning tree containing the most frequently used edges as stated by service selection. *Equal* now performs better with QuLa's weighted average load-balancing because servers closest to the users are prioritized, reducing the network latency for requests compared to the source-based routing approach.

*Greedy* is least affected by the use of a spanning tree; a greedy service selection assigns maximum load to nearby servers and generally requires a minimized set of edges. This greedy selection results in high traffic load on few edges, which allows Kruskal's algorithm to construct a spanning tree containing all used edges. *SA* is able to adapt to network characteristics by using a greedy load distribution when network latency is relatively large, while using a more distributed approach when network latency decreases. This makes *SA* more robust against the penalty of losing edges by using a spanning tree, giving almost similar result for both QuLa weighted average and source-based routing.

We observe that weighted average routing achieves a close approximation of source-based routing when the spanning tree contains all edges used by the service selection. This further indicates that, assuming we obtain a suitable spanning tree, the QuLa weighted average approach approximates the desired benchmark results and enables route aggregation with minimal




(a) (b)

Fig. 5. response times using a static configuration found by *SA* and a dynamic *JSQ* selection for a large topology with sparse connectivity (LS200) running (a) stable at 50% load and (b) at 90% load with small peaks.

forwarding table state.

Because of the in-network load-balancing converging to a greedy load distribution in sparse networks and trees, nearby servers will operate near maximum capacity earlier than more distant servers, possibly lowering the system stability. The convergence to a greedy distribution explains the small performance loss by weighted average routing compared to source-based routing for high load values in Figure 4. However, we observe that *SA* always produces the best response time, even considering the performance loss induced by a spanning tree, illustrated in Figure 4. This further indicates our statement from section V-B that *SA* is the best choice algorithm to perform service selection, regardless of the network characteristics.

### D. STATIC QULA WEIGHTED AVERAGE VS. DYNAMIC JSQ

We investigate to what extent the dynamic selection of *JSQ* can mitigate the performance lost by not considering network latency (cfr. Equation 2). In our simulation setup, service routers know the queue length of each server; when a client sends a request, the first hop service router looks up the least busy server and assigns the request to it. The service request is then forwarded on the lowest-latency path to that server. This setup is a best-case scenario for *JSQ* which we then compare to the static selection results presented in section IV.

Using Figure 4, we compare the average response time achieved through JSQ with the response time of *SA*, which was the best performing static algorithm in section V-C. The load distribution found by *SA* is translated to the forwarding tables with QuLa weighted average, as discussed in section IV. For sparse networks (Figure 4 a and c) we observe that the response time of *SA* is lower than the response time of *JSQ* selection. Sparse networks generally have longer and fewer paths, resulting in a large network latency between client and servers. However, *JSQ* does not consider the network latency and only attempts to minimize the time spent on server. Assume $T_{proc.}(i)$ the time spent on server if we send the request to the least busy server *i* and $T_{proc.}(j)$ the processing time if we send the request to the closest server *j* ($T_{proc.}(i) < T_{proc.}(j)$). $T_{Lat.}(c, n)$ is the network latency between a client *c* and a server *n* ($T_{Lat.}(c, j) < T_{Lat.}(c, i)$). As long as the network latency $T_{Lat.}(c, i) - T_{Lat.}(c, j)$ is larger than the processing time $T_{proc.}(j) - T_{proc.}(i)$, *JSQ* performs worse than *SA*.

Figure 5 shows the response time distribution for both *SA* and *JSQ* under 50% and 90% server load. When the servers run at 50% load (Figure 5 a), a small peak in the expected demand pattern will not heavily affect the server queuing time. As a result, it is not worth sending requests to less loaded but more distant servers using *JSQ*. For 50% load users will experience the lowest response times for *SA*. When servers run at 90% load (Figure 5 b), a small peak in demand can overload the closest server *j* and result in large processing times $T_{proc.}(j) - T_{proc.}(i)$. Although 88% of the users still experience a higher response time with *JSQ* than with *SA*, *JSQ* manages to keep the maximum response time limited while the *SA* distribution has a long tail of users experiencing very large response times due to overloaded servers. *JSQ* was able to keep the servers stable and makes up for the additional network latencies.

However, as illustrated in Figure 4 b and d, dense networks have shorter paths and lower network latency between client and server, making *JSQ* a better choice for service selection in these networks. For sparse networks *JSQ* became a better choice when the expected server load was above 90% while for dense networks this already happens at 50-60% server load.

We conclude that our proposed static configuration technique (cfr. section IV) results in a lower response time as long as the network latency is not negligible compared to the expected server processing times. Although our static algorithm is more complex and requires more information than *JSQ*, it only requires a onetime configuration as long as the expected load conditions do not change.

### E. IN-NETWORK LOAD-BALANCING PERFORMANCE

To avoid routing loops in QuLa weighted routing, requests must be forwarded on edges belonging to a spanning tree, as mentioned in section IV-B. Kruskal's algorithm constructs a minimal spanning tree using an edge metric to find the best set

of edges. In this section we study the effect of this edge metric on the average response time and compare simulation results of two alternative edge metrics.

We propose to use the expected traffic load as edge weight metric, obtained through either source routing or the weighted average variant. The traffic load is a result of the selection process which used Equation 2 as quality metric, thus already considering network latency, server load and the impact from individual clients on the average response time. If expected traffic load on an edge is high, that edge is likely to be heavily used, and should be part of the minimum spanning tree. After inverting the expected traffic load on each edge, Kruskal's algorithm is used to obtain a minimized spanning tree. This guarantees no routing loops can occur while still prioritizing edges which are often used in the load distribution.

*SA* finds a load distribution matrix using the network graph as input, which is then mapped onto the service routing plane using QuLa weighted average. To avoid routing loops, the load distribution matrix is mapped onto the service routers following the constructed spanning tree. A spanning tree narrows the set of paths requests can follow and thus the degree of load-balancing service routers can perform.

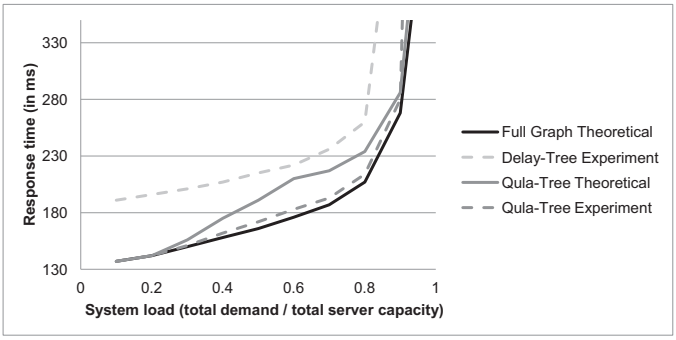



Fig. 6. the influence of a spanning tree on the performance of QuLa weighted average routing in SD50.

To demonstrate the importance of the edge metric used in Kruskal's algorithm, we run simulations using two different spanning trees and compare both results. Next, we show that QuLa weighted average routing is able to perform better than predicted by deviating from the load distribution matrix due to statistical load-balancing on each service router. Both effects are illustrated in Figure 6 for a small network with dense connectivity (SD50). The Full Graph curve represents the expected response time when all graph edges are available during simulation, calculated with Equation 2. The Delay-Tree and Qula-Tree curves illustrate response times when only a subset of edges is available in the graph. We differentiate between the theoretical curves which indicate the expected average response time calculated with Equation 2 and the experimental curves which indicate measured response times during simulation.

First we tackle the importance of the edge metric in Kruskal's algorithm as illustrated in Figure 6; the Delay-Tree experimental curve represents the measured average response time of a spanning tree constructed using the edge latency as weight in Kruskal's algorithm. Our second approach, the (dashed) Qula-Tree experimental curve, uses the service selection result to find the most used edges and prioritizes these edges during the construction of the spanning tree, as described in section IV-B. We observe that the Qula-Tree, which only preserves the links with the highest expected traffic (cfr. service selection result), achieves a substantially better response time than the Delay-Tree and performs almost as well as the theoretical benchmark (which contains all edges).

Next, we observe in Figure 6 that the measured response time of the QuLa-Tree (dashed experimental curve) during simulation is lower than the expected response time (Qula-Tree Theoretical curve). This is due to the in-network load-balancing on service routers which reshape the load distribution matrix at runtime. In section IV-B we explained how QuLa's weighted average approach preserves the amount of traffic on each link while disregarding the source. Without considering source addresses, service routers may distribute a client's demand differently than stated by the load distribution matrix. In a spanning tree requests will pass through more service routers as the traffic is concentrated onto fewer but longer paths. Each service router performs statistical load-balancing, reducing the probability of a request reaching more distant servers as it passes more routers. As a result, QuLa weighted average does not exactly follow the load distribution matrix R(i,j,s). Instead, the traffic is *reshaped and requests stay closer to the users*, while servers receive less requests from more distant clients in the network. Thus, QuLa weighted average load-balancing converges to a greedy load distribution, reducing the penalty of using a spanning tree as fewer edges are used. This explains why the measured average response time (dashed QuLa-Tree experimental curve in Figure 6) is lower than the expected average response time (solid QuLa-Tree Theoretical curve in Figure 6) which was calculated assuming that the load distribution matrix R(i,j,s) is respected at-runtime.

This is illustrated in Figure 7; service selection dictates that client 1 sends 49%, 31% and 20% of its demand to server 1, 2 and 3 respectively. However, due to service routers performing in-network load-balancing, over 80% of its traffic arrives at server 1, which is located closest of all 3 servers to client 1. Service routers load-balance fewer requests from other clients to server 1, respecting the total amount of load generated on each server as explained in section IV-B. This in-network load-balancing is QuLa's natural way of adapting to a suboptimal environment to approach the desired benchmark.

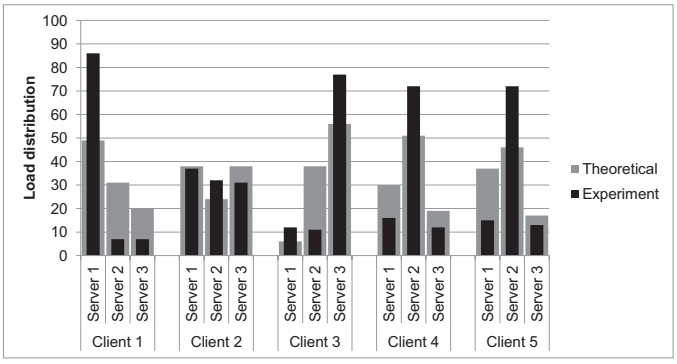



Fig. 7. service routers reshape the load distribution by load-balancing requests at runtime. Results are obtained in a sample topology of SD50.

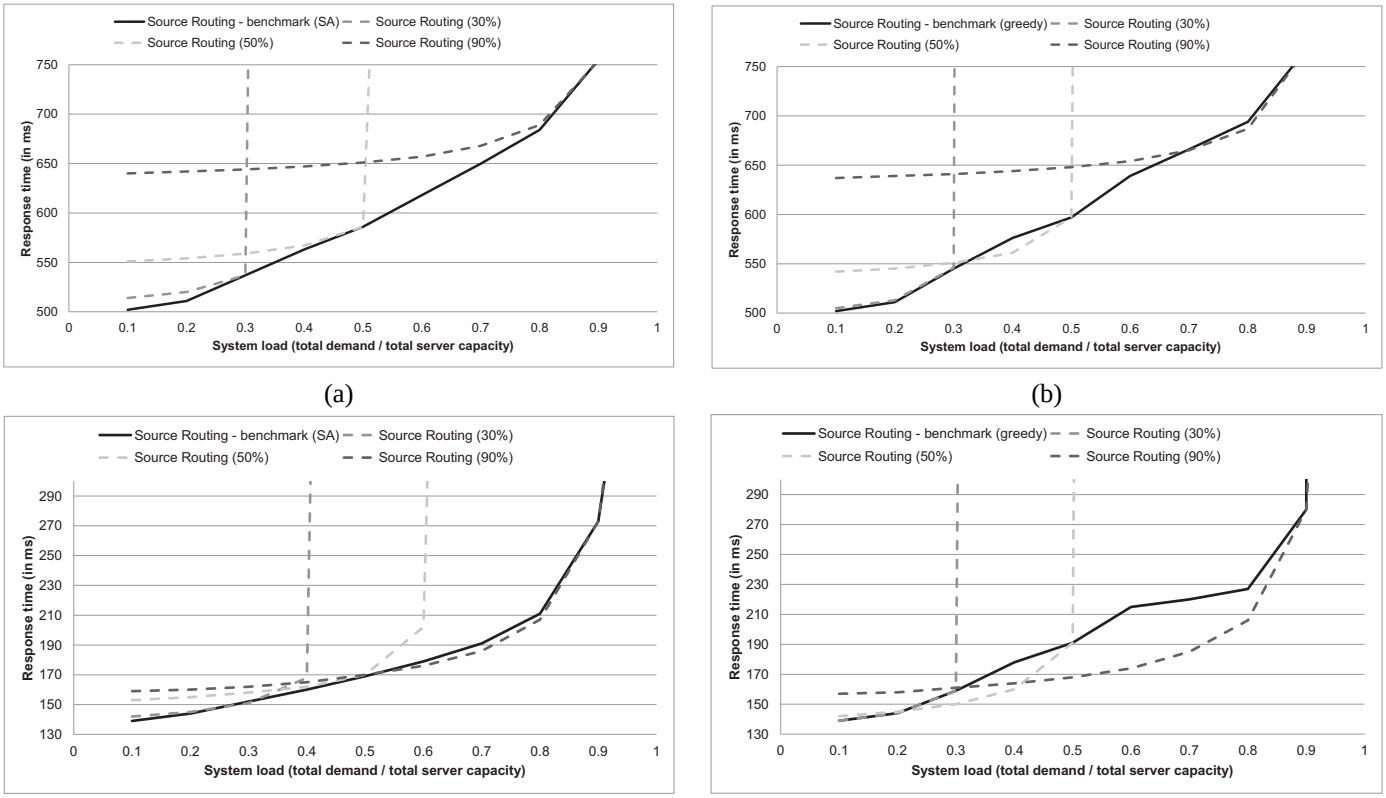



Fig. 8. the importance of measuring the actual demand for (a) *SA* in a sparse network, (b) *Greedy* in a sparse network, (c) *SA* in a dense network and (d) *Greedy* in a dense network.

### *F. ADAPTATION TO DYNAMIC LOAD CONDITIONS*

User demand can vary over time, rendering the current forwarding table configuration less suitable. Reconfiguring the forwarding tables of service routers for the new demand can improve the response time but monitoring demand patterns and system load can be an expensive operation. Therefore we investigated the performance gained by performing service selection using measured demand patterns, compared to service selection which assumes a fixed demand value. We first measure the actual demand and configure the service routers according to the service selection result. These results are illustrated by the solid curves in Figure 8. We then run three new simulations but this time we configure the service routers for fixed demand patterns (30%, 50% and 90% load respectively), while letting the actual demand vary from 10% up to 100%. The average response times obtained by configuring the network for a fixed demand are illustrated by the dashed curves in Figure 8. We evaluated the adaptation to dynamic load conditions for both sparse and dense networks using *SA* and *Greedy*. Figure 8 a and Figure 8 b illustrate the simulation results for a network with sparse connectivity using *SA* and *Greedy* respectively, while Figure 8 c and Figure 8 d are obtained through a network with dense connectivity for *SA* and *Greedy* respectively.

All graphs on Figure 8 show that configuring routers for fixed demand is only feasible when the actual demand does not exceed that pre-determined value; configuring a network for 50% load causes this network to overload when actual load exceeds that value. However, consider the results obtained from a network with dense connectivity (Figure 8 c and d); configuring service routers for 90% load improves the average response time when actual load is lower (50-80%). This shows that *SA* is not always able to find the best solution in a finite time, although it does manage to find a feasible solution (Figure 8 c). Configuring service routers for 90% load when the actual load is lower causes *Greedy* (Figure 8 d) to over-provision and assign more demand than required to distant servers. However, due to the dense nature of the network, the additional network latency and hops to be traversed towards the more distant servers are small, allowing over-provisioning to have positive effects on the average response time. This is essentially what *SA* attempts to do; by increasing the response time for one client (e.g. due to selecting a more distant server) we can reduce the response time of several other clients and possibly reduce the overall average response time. This explains why the *SA* results (Figure 8 c) are similar to the 90% over-provisioning effect on Figure 8 d. However, assigning requests to more distant servers has drastic consequences for the average response time when network latency and hop-count become the more dominant factors. Figure 8 a and Figure 8 b show that when the network latency or average hop-count increases, the average response time considerably increases when the actual demand deviates from the expected value.

We demonstrate the performance loss when only reconfiguring routers when the change in demand exceeds a threshold (e.g. from 30% load to 50% load), compared to measuring the actual

demand during simulation and reconfiguring the routers for every change. Using Figure 8 a we can measure a maximum performance loss of 10% response time by configuring the routers for 90% load when the measured load reaches 50%.

We conclude that accurate monitoring of user demand becomes more important when the cost of using more distant servers increases (due to higher network latency, more hops to traverse, smaller server queue times ...). Therefore, depending on the characteristics of the network, the use of actual demand patterns is a very decisive factor of the system performance.

## VI. CONCLUSION AND FUTURE WORK

In this paper we presented service selection algorithms to seek an optimal distribution of user demand across the deployed service instances, considering both server load and network latency. We described how this selection result can be mapped to the service router forwarding tables using source-based routing and QuLa weighted average routing. Next, we studied the impact on the average response time when service routers load-balance requests without considering the source of a request. To demonstrate the impact of the edge priority metric when constructing a spanning tree, we ran simulations using different metrics to construct the spanning tree. Last, we measured the effect on the response time when configuring routers for actual load and for estimated load values.

As conclusion we can say that the QuLa weighted average routing configuration is able to approximate benchmark results with minimal router state. Upon request arrival, service routers contain all the information needed to load-balance that request to the instance with the fastest response time, without having to query a resolution service (e.g. DNS) or maintain large state. Also, using *SA* we are able to find an optimal service selection independent of the network characteristics, whereas alternative approaches are more sensitive to changing network characteristics.

While we are able to access services with minimal response time using name-based requests, the current service selection algorithm can only handle static demand patterns. When comparing our static configuration to a dynamic assignment through JSQ, we observe that the latter is able to keep the response time stable under peak load but only outperforms a static selection when service times heavily outweigh network latency. The next step in our research is to create scalable dynamic placement and selection algorithms which consider both network and server metrics. Currently, all demand is assumed fixed, and the service selection algorithm searches for the best possible load distribution given a fixed set of instances. When user demand changes, the previous selection is subject to change, requiring reconfiguration of the network. A naive approach runs the static selection algorithm each time user demand changes. This approach is prone to oscillation (over-reacting to changes) and in a worst case scenario the system keeps returning to a previous state, only to repeat this pattern. To avoid circulating previous configurations and to become less sensitive to oscillation, feedback is necessary. We plan on developing a self-learning algorithm and implement control loop feedback. This allows use of prediction algorithms and trend detection, giving the routers a solid foundation to build decisions on.

## VII. ACKNOWLEDGEMENTS

This project was partly funded by the UGent BOF-GOA project "'Autonomic Net- worked Multimedia Systems"', by the FWO-V project "'SPEC: Intelligent SuPer-Elastic Clouds"' and by the 7th Framework Programme of the European Commission through the FUSION project under grant agreement no. 318205.

## REFERENCES

[1] E. Nygren, R. K. Sitaraman, and J. Sun, "The akamai network: a platform for high-performance internet applications," *ACM SIGOPS Operating Systems Review*, vol. 44, no. 3, pp. 2–19, 2010.

[2] A.-J. Su, D. R. Choffnes, A. Kuzmanovic, and F. E. Bustamante, "Drafting behind akamai (travelocity-based detouring)," *ACM SIGCOMM Computer Communication Review*, vol. 36, no. 4, pp. 435–446, 2006.

[3] G. Carofiglio, G. Morabito, L. Muscariello, I. Solis, and M. Varvello, "From content delivery today to information centric networking," *Computer Networks*, vol. 57, no. 16, pp. 3116–3127, 2013.

[4] B. Ahlgren, C. Dannewitz, C. Imbrenda, D. Kutscher, and B. Ohlman, "A survey of information-centric networking," *Communications Magazine, IEEE*, vol. 50, no. 7, pp. 26–36, 2012.

[5] Y. Xu, Y. Li, T. Lin, Z. Wang, W. Niu, H. Tang, and S. Ci, "A novel cache size optimization scheme based on manifold learning in content centric networking," *Journal of Network and Computer Applications*, vol. 37, no. 0, pp. 273 – 281, 2014.

[6] W. K. Chai, D. He, I. Psaras, and G. Pavlou, "Cache less for more in information-centric networks (extended version)," *Computer Communications*, vol. 36, no. 7, pp. 758 – 770, 2013.

[7] C. Dannewitz, M. D'Ambrosio, and V. Vercellone, "Hierarchical dht-based name resolution for information-centric networks," *Computer Communications*, vol. 36, no. 7, pp. 736 – 749, 2013.

[8] L. Zhang, D. Estrin, J. Burke, V. Jacobson, J. D. Thornton, D. K. Smetters, B. Zhang, G. Tsudik, D. Massey, C. Papadopoulos, *et al.*, "Named data networking (ndn) project," *Relatório Técnico NDN-0001, Xerox Palo Alto Research Center-PARC*, 2010.

[9] N. Fotiou, P. Nikander, D. Trossen, and G. C. Polyzos, "Developing information networking further: From psirp to pursuit," in *Broadband Communications, Networks, and Systems*, pp. 1–13, Springer, 2012.

[10] W. K. Chai, N. Wang, I. Psaras, G. Pavlou, C. Wang, G. de Blas, F. Ramon-Salguero, L. Liang, S. Spirou, A. Beben, and E. Hadjioannou, "Curling: Content-ubiquitous resolution and delivery infrastructure for next-generation services," *Communications Magazine, IEEE*, vol. 49, pp. 112–120, March 2011.

[11] T. Koponen, M. Chawla, B.-G. Chun, A. Ermolinskiy, K. H. Kim, S. Shenker, and I. Stoica, "A data-oriented (and beyond) network architecture," *SIGCOMM Comput. Commun. Rev.*, vol. 37, pp. 181–192, Aug. 2007.

[12] C. Dannewitz, D. Kutscher, B. Ohlman, S. Farrell, B. Ahlgren, and H. Karl, "Network of information (netinf): An information-centric networking architecture," *Computer Communications*, vol. 36, no. 7, pp. 721 – 735, 2013.

[13] OGF, "Open cloud computing interface | open standard | open community," 2013.

[14] M. Satyanarayanan, P. Bahl, R. Caceres, and N. Davies, "The case for vm-based cloudlets in mobile computing," *Pervasive Computing, IEEE*, vol. 8, no. 4, pp. 14–23, 2009.

[15] A.-F. Antonescu, A. Gomes, P. Robinson, and T. Braun, "Sla-driven predictive orchestration for distributed cloud-based mobile services," in *Communications Workshops (ICC), 2013 IEEE International Conference on*, pp. 738–743, June 2013.

[16] Q. Yu, "Cloudrec: a framework for personalized service recommendation in the cloud," *Knowledge and Information Systems*, pp. 1–27, 2014.

[17] T. Braun, V. Hilt, M. Hofmann, I. Rimac, M. Steiner, and M. Varvello, "Service-centric networking," in *Communications Workshops (ICC), 2011 IEEE International Conference on*, pp. 1–6, 2011.

[18] M. Freedman, M. Arye, P. Gopalan, S. Ko, E. Nordstrom, J. Rexford, and D. Shue, "Serval: An end-host stack for service-centric networking," in *Proc. USENIX NSDI*, 2012.

[19] M. J. Freedman, M. Arye, P. Gopalan, S. Y. Ko, E. Nordstrom, J. Rexford, and D. Shue, "Service-centric networking with scaffold," *Princeton University, September*, 2010.

[20] K.-W. Lee, B.-J. Ko, and S. Calo, "Adaptive server selection for large scale interactive online games," *Computer Networks*, vol. 49, no. 1, pp. 84–102, 2005.
[21] P. Wendell, J. W. Jiang, M. J. Freedman, and J. Rexford, "Donar: decentralized server selection for cloud services," in *ACM SIGCOMM Computer Communication Review*, vol. 40, pp. 231–242, ACM, 2010.
[22] L. Zhao, Y. Ren, M. Li, and K. Sakurai, "Flexible service selection with user-specific qos support in service-oriented architecture," *Journal of Network and Computer Applications*, vol. 35, no. 3, pp. 962 – 973, 2012. Special Issue on Trusted Computing and Communications.
[23] H. Xu and B. Li, "Joint request mapping and response routing for geo-distributed cloud services," in *INFOCOM, 2013 Proceedings IEEE*, pp. 854–862, IEEE, 2013.
[24] H. A. Alzoubi, S. Lee, M. Rabinovich, O. Spatscheck, and J. Van der Merwe, "Anycast cdns revisited," in *Proceedings of the 17th international conference on World Wide Web*, pp. 277–286, ACM, 2008.
[25] L. Wang, A. Hoque, C. Yi, A. Alyyan, and B. Zhang, "Ospfn: An ospf based routing protocol for named data networking. university of memphis and university of arizona," tech. rep., Tech. Rep, 2012.
[26] C. Li, K. Okamura, and W. Liu, "Ant colony based forwarding method for content-centric networking," in *Advanced Information Networking and Applications Workshops (WAINA), 2013 27th International Conference on*, pp. 306–311, March 2013.
[27] S. Shanbhag, N. Schwan, I. Rimac, and M. Varvello, "Soccer: Services over content-centric routing," in *ACM SIGCOMM Information-Centric Networking (ICN) workshop, Toronto, Canada*, 2011.
[28] E. W. Dijkstra, "A note on two problems in connexion with graphs," *Numerische mathematik*, vol. 1, no. 1, pp. 269–271, 1959.
[29] K. A. Dowsland and C. Reeves, "Modern heuristic techniques for combinatorial problems," *Simulated Annealing. In Reevees, CR, Editor, John Wiley and Sons, NY, USA*, no. 2, 1993.
[30] S. Abimannan, K. Durai, A. Jeyakumar, *et al.*, "Join-the-shortest queue policy in web server farms," *Global Journal of Computer Science and Technology*, vol. 10, no. 4, 2010.
[31] V. Gupta, M. H. Balter, K. Sigman, and W. Whitt, "Analysis of join-the-shortest-queue routing for web server farms," *Performance Evaluation*, vol. 64, no. 9-12, pp. 1062 – 1081, 2007. Performance 2007 26th International Symposium on Computer Performance, Modeling, Measurements, and Evaluation.
[32] J. Kruskal, Joseph B., "On the shortest spanning subtree of a graph and the traveling salesman problem," *Proceedings of the American Mathematical Society*, vol. 7, no. 1, pp. pp. 48–50, 1956.
[33] "The clouds lab: Flagship projects - gridbus and cloudbus," 2013.
[34] "Brite: Boston university representative internet topology generator," 2013.
[35] "ilab.t virtual wall | internet based communication networks and services," 2013.

**Piet Smet** received his M.Sc degree in Informatics (2012) from Hogeschool Ghent, Belgium. He wrote a thesis on the development of a large-data social media game during his Masters degree. In August 2012 he started a Ph.D. degree on 'Service-Centric Networking' at Ghent University, supervised by B.Dhoedt and P. Simoens. Currently, Piet Smet is active as a researcher on the European research project H2020 FUSION.

**Pieter Simoens** received his M.Sc. degree in Electronic Engineering (2005) and Ph.D. degree (2011) from the Ghent University, Belgium. During his Ph.D. research, he was funded by the Fund for Scientific Research Flanders (FWO-V). In 2012, he was a visiting researcher at the School of Computer Science of Carnegie Mellon University, USA. Currently, he is assistant professor affiliated with the Department of Information Technology of the Ghent University and with iMinds. His main research interests include mobile cloud offloading, service-oriented networking, edge/fog computing paradigms, and service engineering for advanced mobile applications. In these fields, he is author and co-author of more than 70 papers published in international journals or in the proceedings of international conferences. He has also been involved in several national and European research projects (FP6 MUSE, FP7 MobiThin, H2020 FUSION).

**Bart Dhoedt** received a Masters degree in Electrotechnical Engineering (1990) from Ghent University. His research, addressing the use of micro-optics to realize parallel free space optical interconnects, resulted in a Ph.D. degree in 1995. After a 2-year post-doc in opto-electronics, he became Professor at the Department of Information Technology. He is author or co-author of more than 300 publications in international journals or conference proceedings.